Общи задачи

Тодор Дуков

Примери за задачи с числа

Искаме да напишем алгоритъм, който при подадена двойка от естествени числа $\langle a, b\rangle$, където $b \leq a$, да върне $\gcd(a, b)$ т.е. това число $d \in \mathbb{N}$, за което:

- $d \mid a$ и $d \mid b$
- ако $d_1 \mid a$ и $d_1 \mid b$, то $d_1 \mid d$

За целта ще покажем, че за всяко $a, b, q, r \in \mathbb{N}$, където $0 < b \leq a$, $a = bq + r$ и $r \in \{0, \ldots, b-1\}$, е изпълнено, че:

$$\gcd(a, b) = \gcd(b, r)$$

Първо имаме, че $\gcd(a, b) \mid a$ и $\gcd(a, b) \mid b$, откъдето понеже $a = bq+r$, имаме $\gcd(a, b) \mid r$. Тогава $\gcd(a, b) \mid \gcd(b, r)$. От друга страна $\gcd(b, r) \mid b$ и $\gcd(b, r) \mid r$, откъдето $\gcd(b, r) \mid bq + r = a$. Така $\gcd(b, r) \mid \gcd(a, b)$. Накрая можем да заключим $\gcd(a, b) = \gcd(b, r)$ от $\gcd(a, b) \mid \gcd(b, r)$ и $\gcd(b, r) \mid \gcd(a, b)$.
На базата на това наблюдение се получава следният алгоритъм (кръстен на Евклид):

```
int euclid(int a, int b) // a >= b
{
    if (b == 0)
        return a;

    return euclid(b, a % b);
}
```

Ще покажем с индукция относно $\mathtt{b}$, че за всяко $\mathtt{a} \geq \mathtt{b}$, е изпълнено, че:

$$\mathtt{euclid}(\mathtt{a}, \mathtt{b}) = \mathtt{gcd}(\mathtt{a}, \mathtt{b})$$

- В базовия случай имаме $\mathtt{euclid}(\mathtt{a}, 0) = \mathtt{a} = \mathtt{gcd}(\mathtt{a}, 0)$
- Нека $\mathtt{b} > 0$ и $\mathtt{a} = \mathtt{bq} + \mathtt{r}$ за някои $\mathtt{r} \in \{0, \ldots, \mathtt{b} - 1\}$ и $\mathtt{q}$ и нека твърдението е изпълнено за всяко $\mathtt{b}' < \mathtt{b}$. Тогава:

$$\mathtt{euclid}(\mathtt{a}, \mathtt{b}) = \mathtt{euclid}(\mathtt{b}, \mathtt{r}) \overset{\text{(ИП)}}{=} \mathtt{gcd}(\mathtt{b}, \mathtt{r}) = \mathtt{gcd}(\mathtt{a}, \mathtt{b})$$

Алгоритъмът терминира, понеже управляващият параметър $\mathtt{b}$ винаги намаля, докато не стане $\mathtt{0}$. На пръсти ще покажем, че алгоритъмът има сложност по време и памет $O(\log(a))$. Нека видим, че ако $a > b$, то $\mathrm{mod}(a, b) < \frac{a}{2}$:

1 сл. ако $b \leq \frac{a}{2}$, то $\mathrm{mod}(a, b) < \frac{a}{2}$ и сме готови

2 сл. ако пък $b > \frac{a}{2}$, то тогава $a - b < \frac{a}{2}$, откъдето $\mathrm{mod}(a, b) = \mathrm{mod}(a - b, b) = a - b < \frac{a}{2}$

Тогава през на всеки две стъпки на алгоритъма от вход $\langle a,\ b\rangle$, отиваме до вход $\langle \mathrm{mod}(a,\ b),\ \mathrm{mod}(b,\ \mathrm{mod}(a,\ b))\rangle$ и левият аргумент става поне два пъти по-малък. Тъй като левият аргумент е горна граница за десния, то той също ще намаля рязко. Дълбочината на дървото на рекурсията зависи само от броя на рекурсивните извиквания. Понеже те са логаритмично много и всяко едно от тях заема константна памет, сложността по памет е също $O(\log(a))$.
Втората задача за числа е скрита в задача за масиви:
Имаме един масив $\mathtt{A}[1 \ldots \mathtt{n}]$, който съдържа всички числа от $\mathtt{1}$ до $\mathtt{n}$, с изключение на едно от тях, което е заместено с друго число от $\mathtt{1}$ до $\mathtt{n}$. Искаме да напишем колкото се може по-бърз алгоритъм, който при вход такъв масив $\mathtt{A}$ връща наредената двойка $\langle \mathtt{dup}, \mathtt{miss}\rangle$ от дупликата и липсващото число. Оказва се, че тази задача може да се реши за линейно време и константна памет. За целта трябва да се забележат следните факти:

- $\mathtt{dup} - \mathtt{miss} = \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{A}[\mathtt{i}] - \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{i} = \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{A}[\mathtt{i}] - \frac{\mathtt{n}(\mathtt{n}+1)}{2} =: \mathtt{S}_{1,\mathtt{A}}$
- $(\mathtt{dup} + \mathtt{miss})(\mathtt{dup} - \mathtt{miss}) = \mathtt{dup}^2 - \mathtt{miss}^2 = \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{A}[\mathtt{i}]^2 - \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{i}^2 = \sum\limits_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{A}[\mathtt{i}]^2 - \frac{\mathtt{n}(\mathtt{n}+1)(2\mathtt{n}+1)}{6} =: \mathtt{S}_{2,\mathtt{A}}$

От тях получаваме следната система от уравнения:

$$\left|\begin{array}{l} \mathtt{dup} - \mathtt{miss} = \mathtt{S}_{\mathtt{1},\mathtt{A}} \\ \mathtt{dup} + \mathtt{miss} = \frac{\mathtt{S}_{\mathtt{2},\mathtt{A}}}{\mathtt{S}_{\mathtt{1},\mathtt{A}}} \end{array}\right.$$

Най-сложното, което трябва да направим, е да пресметнем $\mathtt{S}_{\mathtt{1},\mathtt{A}}$ и $\mathtt{S}_{\mathtt{2},\mathtt{A}}$:

```
1  pair<int, int> find_missing_and_duplicate(int *arr, int n)
2  {
3      int sum = (n * (n + 1)) / 2;
4      int sq_sum = (n * (n + 1) * (2 * n + 1)) / 6;
5
6      int arr_sum = 0;
7      int arr_sq_sum = 0;
8
9      for (int i = 0; i < n; ++i)
10     {
11         arr_sum += arr[i];
12         arr_sq_sum += arr[i] * arr[i];
13     }
14
15     int dup = ((arr_sum - sum) + (arr_sq_sum - sq_sum) / (arr_sum - sum)) / 2; // (S1 + S2 / S1) / 2
16     int miss = dup - arr_sum + sum; // dup - S1
17
18     return {dup, miss};
19 }
```

Остава само да се докаже следното твърдение:

Инвариант. *При всяко достигане на проверката за край на цикъла на ред* 9 *имаме, че:*

$$\mathtt{arr\_sum} = \sum_{\mathtt{k}=0}^{\mathtt{i}-1} \mathtt{arr}[\mathtt{k}] \text{ и } \mathtt{arr\_sq\_sum} = \sum_{\mathtt{k}=0}^{\mathtt{i}-1} \mathtt{arr}[\mathtt{k}]^2$$

База. При първото достигане имаме, че:

- $\mathtt{arr\_sum} = 0 = \sum_{\mathtt{k}=0}^{0-1} \mathtt{arr}[\mathtt{k}]$
- $\mathtt{arr\_sq\_sum} = 0 = \sum_{\mathtt{k}=0}^{0-1} \mathtt{arr}[\mathtt{k}]^2$

Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава влизайки в тялото на цикъла:

- $\mathtt{arr\_sum}$ става $\mathtt{arr\_sum} + \mathtt{arr}[\mathtt{i}] \overset{(\text{ИП})}{=} \sum_{\mathtt{k}=0}^{\mathtt{i}-1} \mathtt{arr}[\mathtt{k}] + \mathtt{arr}[\mathtt{i}] = \sum_{\mathtt{k}=0}^{\mathtt{i}} \mathtt{arr}[\mathtt{k}] = \sum_{\mathtt{k}=0}^{\mathtt{i}_{\mathtt{new}}-1} \mathtt{arr}[\mathtt{k}]$
- $\mathtt{arr\_sq\_sum}$ става $\mathtt{arr\_sq\_sum} + \mathtt{arr}[\mathtt{i}]^2 \overset{(\text{ИП})}{=} \sum_{\mathtt{k}=0}^{\mathtt{i}-1} \mathtt{arr}[\mathtt{k}]^2 + \mathtt{arr}[\mathtt{i}]^2 = \sum_{\mathtt{k}=0}^{\mathtt{i}} \mathtt{arr}[\mathtt{k}]^2 = \sum_{\mathtt{k}=0}^{\mathtt{i}_{\mathtt{new}}-1} \mathtt{arr}[\mathtt{k}]^2$

Терминация. В последното достигане на проверката за край на цикъла имаме, че $\mathtt{i} = \mathtt{n}$, откъдето:

$$\mathtt{arr\_sum} = \sum_{\mathtt{i}=0}^{\mathtt{n}-1} \mathtt{arr}[\mathtt{i}] \text{ и } \mathtt{arr\_sq\_sum} = \sum_{\mathtt{i}=0}^{\mathtt{n}-1} \mathtt{arr}[\mathtt{i}]^2$$

Имайки това твърдение, остава само да се отбележи, че `dup` и `miss` наистина са решения на горната система от уравнения (със съответните преиндексирания). Накрая нека за пълнота с индукция по $n \in \mathbb{N}$ покажем, че:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \text{ и } \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

- $\sum_{i=0}^{0} i = 0 = \frac{0(0+1)}{2}$ и $\sum_{i=0}^{0} i^2 = \frac{0(0+1)(2\cdot 0+1)}{6}$ ✓
- $\sum_{i=0}^{n+1} i = \sum_{i=0}^{n} i + (n+1) \overset{(\text{ИП})}{=} \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$
- $\sum_{i=0}^{n+1} i^2 = \sum_{i=0}^{n} i^2 + (n+1)^2 \overset{(\text{ИП})}{=} \frac{n(n+1)(2n+1)}{6} + (n+1)^2 = \frac{n(n+1)(2n+1)}{6} + \frac{6(n+1)^2}{6} = \frac{(n+1)(2n^2+n+6n+6)}{6} = \frac{(n+1)(n+2)(2(n+1)+1)}{6}$

Задачи

Задача 1. Да се напише алгоритъм `find_primes(n)`, който приема естествено число $\mathtt{n}$ и връща булев масив $\mathtt{P}[2 \dots \mathtt{n}]$, за който е изпълнено, че за всяко $2 \leq \mathtt{i} \leq \mathtt{n}$:

$$\mathtt{P}[\mathtt{i}] \text{ е истина} \iff \mathtt{i} \text{ е просто}$$

След това да се докаже неговата коректност и да се изследва сложността му по време и памет.

Задача 2. Масив на Монж ще наричаме всеки двумерен масив $\mathtt{A}[1 \dots \mathtt{n}, 1 \dots \mathtt{m}]$ от естествени числа, за който:

$$\mathtt{A}[\mathtt{p}, \mathtt{q}] + \mathtt{A}[\mathtt{s}, \mathtt{t}] \leq \mathtt{A}[\mathtt{p}, \mathtt{t}] + \mathtt{A}[\mathtt{s}, \mathtt{q}] \text{ за всички } 1 \leq \mathtt{p}, \mathtt{s} \leq \mathtt{n} \text{ и } 1 \leq \mathtt{q}, \mathtt{t} \leq \mathtt{n}$$

Да се напише алгоритъм `test_mongeness(A)` със линейна сложност, който приема двумерен масив $\mathtt{A}[1 \dots \mathtt{n}, 1 \dots \mathtt{m}]$ и проверява дали е масив на Монж. След това да се докаже неговата коректност и да се изследва сложността му по време и памет.

Задача 3. Да се напише колкото се може по-бърз алгоритъм `split(A)`, който приема масив $\mathtt{A}[1 \dots \mathtt{n}]$ от цели числа и го подрежда така, че всички отрицателни числа да са вляво от всички неотрицателни. След това да се докаже неговата коректност и да се изследва сложността му по време и памет.

Задача 4. Да се напише колкото се може по-бърз алгоритъм `find_products(A)`, който приема масив $\mathtt{A}[1 \dots \mathtt{n}]$ от цели числа и връща масив $\mathtt{B}[1 \dots \mathtt{n}]$, такъв че за всяко $1 \leq \mathtt{i} \leq \mathtt{n}$:

$$\mathtt{B}[\mathtt{i}] = \prod_{\substack{\mathtt{k}=1 \\ \mathtt{k} \neq \mathtt{i}}}^{\mathtt{n}} \mathtt{A}[\mathtt{i}]$$

След това да се докаже неговата коректност и да се изследва сложността му по време и памет.

Задача 5. Да се напише колкото се може по-бърз алгоритъм `find_max_product_pair(A)`, който приема масив $\mathtt{A}[1 \dots \mathtt{n}]$ от цели числа и връща двойка $\langle \mathtt{i}, \mathtt{j} \rangle$ от различни индекси, за които $\mathtt{A}[\mathtt{i}] * \mathtt{A}[\mathtt{j}]$ е максимално. След това да се докаже неговата коректност и да се изследва сложността му по време и памет.

Задача 6. Даден е следният алгоритъм:

```
int alg(int *arr, int n)
{
    bool x = true, y = true;

    for (int i = 0; i < n - 1; ++i)
    {
        if (x && arr[i] > arr[i + 1])
            x = false;
        else if (!x && arr[i] < arr[i + 1])
            y = false;
    }

    return y;
}
```

Какво връща `alg(arr, n)` където `arr` е масив от цели числа с дължина `n`? Обосновете отговора си.

Задача 7. Да се напише колкото се може по-бърз алгоритъм, който при подадено крайно множество от точки $P \subseteq \mathbb{Z} \times \mathbb{Z}$, намира $\max\{|x_1 - x_2| + |y_1 - y_2| \mid (x_1, y_1), (x_2, y_2) \in P\}$. След това да се докаже неговата коректност и да се изследва сложността му по време и памет.